EXHIBIT G

REAL-TIME APPROXIMATIONS OF THE RENDERING EQUATION

Anthony Ambrus[*]

School of Electronics and Computer Science
University of Southampton

ABSTRACT

Physically correct rendering is not possible in real-time due to the complexities of light-scene interaction and their computational cost. Instead, images are rendered using approximations of varying levels of realism. This report reviews the major seminal methods in recent years, presenting material effects including parallax mapping and sub-surface scattering, comparing advanced shadowing methods and examining current real-time global illumination techniques. The results of a survey on the relative perceptual importance of the reviewed algorithms is presented, concluding that global illumination methods have the most impact, followed by material effects and then shadowing.

Index Terms— real-time, rendering, algorithms, global illumination, shadows, materials

1. INTRODUCTION

One of the fundamental concerns of computer graphics is realistic lighting. In real-time rendering, this problem poses additional challenges; algorithms must also be efficient enough to provide interactive frame rates. The conflict between realism and speed has led to the exploration of methods approximating the rendering equation. This paper reviews the theory and implementation behind several of the most significant developments in realistic lighting in recent history.

The remainder of this paper is structured as follows. We examine the problem of rendering realistic images in real-time in section 2. Sections 3 and 4 review a selection of direct and global illumination techniques (respectively), which arguably contribute the most to current real-time rendering; section 5 considers a range of effects caused by the limitations of optical sensing equipment, but which are necessary to providing a realistic rendition of a scene.

In section 6 we submit the results of a comparison of the contribution importance of a number of the algorithms presented. Section 7 concludes.

2. MOTIVATION AND BACKGROUND

Lighting is a very important tool in portraying scenes. It can be used to create a 3D effect by separating foreground and background, or it can merge the two to create a flat 2D effect. Correctly used lighting can be used to set an emotional mood and to influence the viewer. Spatial information can be extracted from a scene by the effect of lighting, and its counterpoint shadowing. The direction and intensity of the light source, the surface material and the geometry of shadow receivers are just a few issues that are discernable purely by the interplay of light over objects.

In order to illuminate a scene correctly, however, to accurately address these issues, not only must direct lighting be taken into account but also the complex inter-reflection between surfaces. The model representing this situation is known as global illumination and is expressed with the following equilibrium equation:

$$L(x,\vec{\omega}_o) = L_e(x,\vec{\omega}_o) + \int_S f_r(x,\vec{\omega}_i \rightarrow \vec{\omega}_o) L(x',\vec{\omega}_i) G(x,x') V(x,x') d\omega_i$$

This can be broken down into constituent functions:

$L(x,\vec{\omega}_o)$ = the intensity reflected from position x in direction ω_o

$L_e(x,\vec{\omega}_o)$ = the light emitted by the object itself

$f_r(x,\vec{\omega}_i \rightarrow \vec{\omega}_o)$ = the BRDF of the surface at point x, transforming incoming light ω_i to reflected light ω_o

$L(x',\vec{\omega}_i)$ = light from x' on another object arriving along ω_i

$G(x,x')$ = the geometric relationship between x and x'

$V(x,x')$ = a visibility test, returns 1 if x can see x', 0 otherwise

This is known as the rendering equation [1], derived solely by physical effects. As such it is the gold standard of realistic lighting. However, the formulation is by no means a real-time algorithm; it is an integral over a hemisphere and the desired light intensity function, L, appears on both sides of the equation.

To maintain interactive frame rates, approximations to the rendering equation must be used. Standard illumination techniques, used by current real-time graphics APIs OpenGL and DirectX [2], use a direct illumination model which ignores most terms of the rendering equation. However, as computing power increases, rendering methods require less approximation and techniques previously only leveraged by offline renderers are now being brought to bear in real-time applications: shadow techniques are now widespread to reintroduce the visibility

[*] e-mail: aja303@zepler.net

component of the rendering equation. Luxo Jr., the first computer-animated short film, utilised two 2048x2048 shadow maps to cast realistic shadows. The next generation of games, regularly use multiple 1024x1024 maps, producing soft shadows in real-time.

Material reflectance properties, known as BRDFs (bidirectional reflectance distribution functions [3]), are significantly more accurate, increasing obtained from image-space analysis of real materials. This is most evident in renders of human faces, possibly the greatest test of realism as a significant portion of the brain is involved in recognising the human face. Along with the inevitable training since birth, this has made us experts in noticing even small irregularities, noticing features in the order of ~100 microns. In Figure 1 you can see the progress made in rendering believable faces, modern methods including the translucency of skin by the use of sub-surface scattering (see section 3.2.3).

Figure 1: Believable rendering of human faces: from *Tin Toy* (Pixar, 1998) to *Matrix Reloaded* (ESC Entertainment, 2003).

The challenges to render realistic objects can be divided into three categories: general lighting environments, which account for the integration of lighting components from all directions (*light integration complexity*); the bouncing and occluding effects, such as shadows, due to intervening geometry between the source and receiver (*light transport complexity*) and the modelling of complex, spatially-varying materials, stored as *bidirectional reflectance distribution functions* (*BRDF complexity*). Varying levels of realism can be achieved by the use of a subset of these components; the standard illumination model ignores the complexities of light transport and represents surfaces as a combination of perfect diffuse and specular reflectors, a special case of BRDFs.

Figure 2: Realism of real-time models in *Gears of War* (2007)

Figure 2 illustrates the level currently achievable in real-time with mainstream hardware. The introduction of the next-generation of consoles, specifically the Xbox 360 and Playstation 3, has updated the definition of mainstream hardware, but the novelty of the platforms has precluded developers from utilising the maximum capabilities of the systems. Similarly, a slew of papers have been published in recent years challenging the limitations of real-time graphics, the consequences of which have yet to be complete felt. Over the coming years, as research and experience with the platforms and techniques grow, methods that currently perform at interactive (~15fps) or lower frame rates will inevitably be relocated to real-time scope.

3. DIRECT ILLUMINATION

Direct illumination encompasses the set of algorithms that only take into account light that comes directly from a light source, ignoring the effects of subsequent surface reflections. This reduces the rendering equation from an integral over a hemisphere to a summation of the incident light to an arbitrary number of point lights, which can be efficiently computed in hardware:

$$L(x,\vec{\omega}_o) = L_e(x,\vec{\omega}_o) + \sum_{n=0}^{lights} f_r(x,\vec{\omega}_i \rightarrow \vec{\omega}_o) L(n) G(x,n) V(x,n)$$

Light integration is the computation of the energy received by a surface. It is denoted in the above equation by the $f_r(x,\vec{\omega}_i \rightarrow \vec{\omega}_o)L(n)G(x,n)$ functions, where $L(n)$ accounts for the light's diffuse and specular properties and $G(x,n)$ is the energy attenuation suffered due to the position and direction of the light relative to the surface. The visibility test, $V(x,n)$, is not part of the fixed-function pipeline of current real-time APIs, due to the inherently high-level knowledge the function must access and its situation-specific nature.

3.1. Light Integration

The lighting equations used by OpenGL and DirectX in the fixed-function pipeline utilise the Phong reflection model to determine the colour and intensity of light reflected from a surface. It is a *phenomenological* model, in that it produces realistic effects but is not physically accurate, disregarding reciprocality and conservation of energy.

The Phong reflection model separates surface reflection into three subcomponents: specular (i_s), diffuse (i_d), and ambient reflection (i_a), the ambient term included to compensate for the lack of inter-reflected light. The BRDF of the material is also assumed to be only constituted of ambient, diffuse and specular component (denoted by k_a, k_d and k_s respectively):

$$I = k_e + k_a i_a + \sum_{n=0}^{lights} k_d (L \cdot N) i_d + k_s (R \cdot V)^{\alpha} i_s$$

In practice, the Blinn-Phong shading model is used in standard illlumination, as it trades visual precision for computing efficiency. In Phong shading, the angle between the light and the viewer on a surface must be continually recalculated, whereas Blinn's model only needs to be recalculated once per frame when dealing with lights and observers at infinite distances [4] (the default setup for OpenGL). Additionally, whilst only an approximation of the Phong model, the Blinn-Phong modification produces more accurate models of empirically derived BRDFs for many material types [5].

Lighting calculations are typically expensive, and are avoided as much as possible during rendering either by pre-calculating the lighting environment (light-maps) or by controlling the unit of lighting. Polygon-based renderers can thus perform lighting on a per-polygon, vertex or pixel basis.

Per-polygon lighting is called *flat shading*, in which just one colour is computed and used to shade a triangle. Silhouette effects render correctly, but lighting based directly on the planar approximation generates unrealistic results with visible discontinuities along polygon edges. This faceted look can be diminished by reducing the size of the triangles used to portray the object. However, number of illumination calculations is proportional to the triangle count, greatly impacting the rendering speed.

Gouraud shading reduces the discontinuities along boundaries by linearly interpolating the colour between vertices. Objects with high specular components have inappropriately shaped highlights, as they depend on the shape of the polygon used to approximate the curved surface. In the case where highlights are located between vertices, the entire highlight may be missed completely. This can cause significant frame to frame discontinuities when shading surfaces in motion, due to the changing properties of polygons describing the surface.

Phong shading improves on Gouraud shading by interpolating the surface normal between vertices (Figure 3), as opposed to the vertex colour. The interpolation is identical, except that the surface normals are renormalized to unit length on a per-pixel basis. As a result, this is more computationally expensive and is used sparingly.

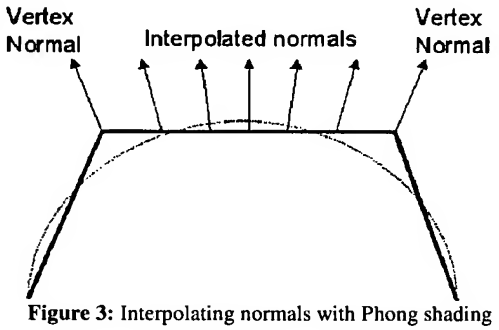


Figure 3: Interpolating normals with Phong shading

Figure 4: Flat, Gouraud and Phong Shading models applied to a cone, from [6].

3.2. Material Definitions

Until as recently as 2000 [7], developers were limited to creating materials that fitted into the fixed nature of the rendering pipeline. However, the maturation of graphics cards into more abstract parallel processing units reduced these restraints, allowing programmable pipelines through the use of vertex and pixel shaders. In the short term this has led to effects like normal mapping and per-pixel lighting, but looking to the cutting edge we can see complex materials properties including subsurface scattering and multi-depth layering being harnessed to create the next generation of realistic materials.

Programmable-function pipelines have made other physically based reflection models such as Oren-Nayar [8] and Cook-Torrance [3] possible to implement simply in shaders. The effects of complex microfaceted materials can be more accurately represented, such as the Ashikhmin [9] models for anisotropic[2] materials. However, these are not currently in widespread use due to the small subset of materials with non-isotropic properties and the increase in computational complexity for relatively small enhancement in realism. Instead, the focus for next generation materials is currently on emulating micro-geometry; acting like the surface has the resolution of the texture. This section examines some of the more important material effects for next generation graphics.

3.2.1. Normal Mapping

Many real-world materials, such as rusted metal and brick, are rough in appearance, caused by the small-scale structures on their surface. The fluctuations in height can be simulated by planar micro-facets as a finely tessellated mesh, but for performance reasons it is usually not practical for large models. Normal mapping, illustrated in Figure 5, allows the approximation of bumpy surfaces without increasing the geometric complexity of a model. This is achieved by storing the perturbed surface normals in a *normal map* texture that can be sampled on a per-pixel basis to vary the illumination of a surface.

[2] Anisotropic materials have skewed reflectance along one axis.

Figure 5: Normal mapping adds per-pixel relief shading to represent rough surfaces by perturbing surface normals.

Multiple coordinate systems are utilised in 3D graphics, including object space, world space and eye space. Normal mapping relies on *texture space*, which is related to the texture coordinates associated with each vertex (see appendix, section 9.1). Conceptually, the normal mapping algorithm proceeds as follows:

1. The precomputed surface normal and tangent for a vertex are passed to the algorithm, from which the texture space coordinate system can be derived
2. A perturbation vector is sampled from the normal map and transformed by the texture space matrix to produce the perturbed normal in the same coordinate system as the surface normal.
3. The perturbed normal is then be substituted for the original normal in the lighting equation to determine the colour at each pixel.

Normal maps are the differentials of height maps, denoting the gradient between pixels due to change in height values. The technique relies on the fact that surfaces with relatively small height variation do not demonstrate noticeable parallax effects. However, normal maps obtained from height maps with large variance produce flat, unrealistic results when viewed at an angle due to the lack of motion parallax. As a result, normal mapping is only usually used for model detail. In situations such as rendering a cobblestone floor, *parallax mapping* can be used as a more accurate simulation.

3.2.2. Parallax Mapping

Unlike normal mapping, parallax mapping [10] accounts for displacement due to differences in height as fragment shader operation. Most textures represent an overhead image of a real surface and as such the texture coordinate for the point on the polygon directly underneath the eye does not need to be adjusted. As the eye moves away from the perpendicular, parallax mapping nudges the texture coordinates for areas of high relief toward the eye, which correspondingly causes those same parts of texture itself to retreat from the eye. Note that the warped mesh in Figure 6 is only approximate; in reality the texture coordinate is modified independently on a per-pixel level, and each fragment is given its own interpolated value of the view vector.

Further to the requirements of standard texture mapping (or normal mapping), two additional inputs are needed by a fragment shader implementing parallax mapping: a height function and the location of the viewer or camera. The height function is nearly always represented by a texture map and can be provided to the shaders encoded as the alpha channel of either the material's colour or normal textures. The view vector, however, must be expressed in *texture space* as described in appendix 9.1.

The actual method is somewhat counter-intuitive. Rather than find the intersection ***B*** between the ***eye*** vector and the height map, which would require some form of ray-tracing, the height value at the intersection T_o of the eye vector and the polygon is used. This value is scaled by the gradient of the eye vector:

$$T_n = T_o + height(T_o) \times \frac{eye_{(x,y)}}{eye_z}$$

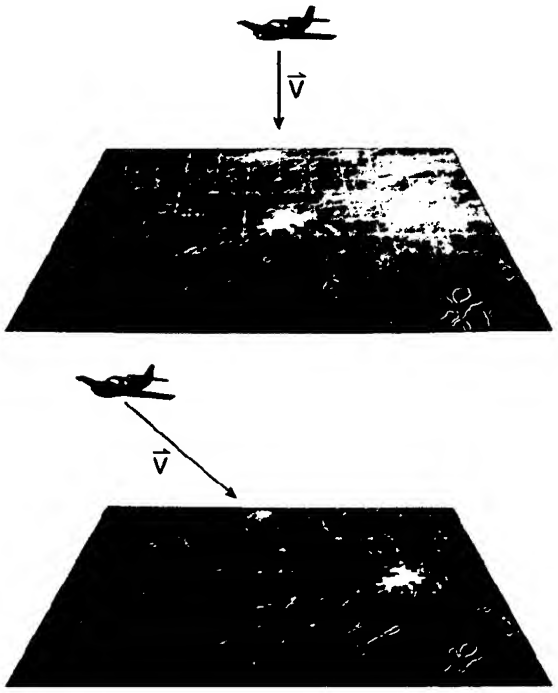


Figure 6: Displacement effects from viewing at texture from an angle using parallax mapping.

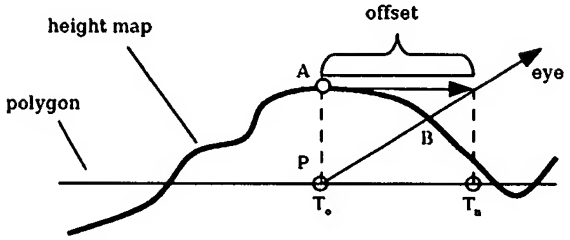


Figure 7: Geometry of texture coordinate shift. Note how the adjusted texture coordinate, T_n, overshoots the correct intersection B.

So, as the observer moves away from the parallel, the new texture coordinate is shifted proportional to the height at the original. The effect of this is to leave low points in the same location, and nudge high points away from the viewer to a location that *more closely approximates* the intersection of the view vector with the height map, though it may over or undershoot as illustrated in Figure 7.

Whilst parallax mapping can produce convincing results for faces relatively perpendicular to the view (Figure 8), the method deteriorates at acute viewing angles for a number of reasons. Firstly, geometry is not being modified by the technique; low polygonal detail will thus be revealed in profile. Additionally, parallax mapping cannot handle situation in which sections of the texture are occluded by others. However, perhaps the most significant issue is caused by high-frequency detail in the texture. At glancing angles, steep gradients in the height map can cause artefacts, known as *texture swim*, in which the adjusted texture coordinates appear overcorrected.

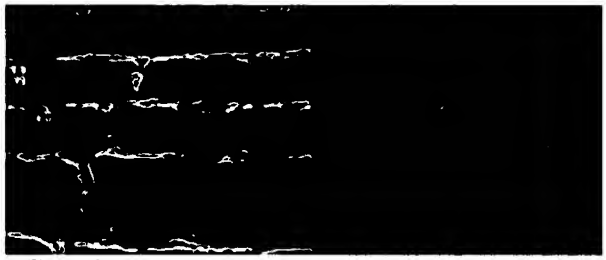


Figure 8: Illusion of depth gained from parallax displacement.

Texture swim can be alleviated by the use of *offset limiting* [11], removing the 1/z term in the parallax equation, resulting in the equation:

$$T_n = T_o + height(T_o) \times eye_{\{x,y\}}$$

Removing the z division limits the texture offset to a maximum distance $height(T_o)$ along both axes, greatly reducing overcorrection artefacts. At steep angles, the modified function approximates the original, as z is close to 1.0. As the angle becomes more acute, the x and y components are not mitigated by the z value, causing the coordinate offset amount to approach 0, "flattening" the surface. While this destroys the parallax effect at glancing angles, it is far less conspicuous than texture swim.

Several extensions have been made to parallax mapping, including *steep parallax maps* [12] to better account for texture swim and self-shadowing; and *distance mapping* [13], a ray-tracing technique that guarantees correct occlusion and maintains "depth" at acute viewing angles.

Parallax mapping is becoming more of a mainstream technique over recent years, its speed the major factor in its adoption. However, as GPU power increases and restrictions on shader length are removed [14], derivatives of parallax mapping, specifically distance mapping, that more accurately model complex self-occluding behaviour will likely takeover from this seminal method.

3.2.3. Sub-surface scattering

Most materials used in real-time graphics today only account for the interaction of light at the surface of an object. In reality, many materials are slightly translucent: light enters the surface; is absorbed, scattered and reemitted - potentially at a different point. Skin is a good case in point; only about 6% of reflectance is direct, 94% is from subsurface scattering [15].

An inherent property of translucent materials is absorption. The farther through the material light travels, the greater the proportion absorbed. To simulate this effect, a measure of the distance light has travelled through the material must be obtained.

One method of estimating this distance is to use depth maps [16], in a manner similar to shadow mapping (see section 3.3.2). The scene is rendered from the light's point of view into a depth map, so that the distance to the nearest surface is stored. The depth map is then projected onto it using standard projective texture mapping and the scene re-rendered. In this pass, when shading a given point, the distance from the light at the point the ray entered the surface can be obtained by a simple texture lookup. By subtracting this value from the point the ray exited the object we can gather an estimate of the distance the light has travelled through the object.

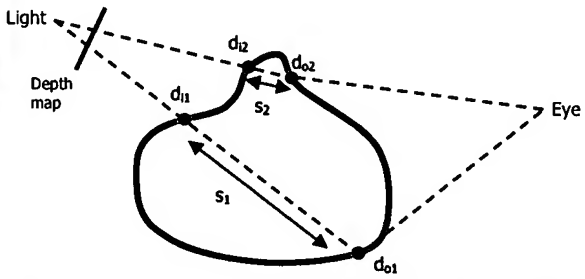


Figure 9: Calculating the distance light has travelled through an object using depth maps.

The measure of distance obtained by this method can be used in several ways. One such way is to use it to index directly into an artist created 1D texture that falls off exponentially with distance. This approach, combined with other more traditional lighting models, allows the creation of different materials such as marble, jade or wax.

Potentially, problems can arise if models are not complex, but *depth peeling* [17] can be used to avoid the issue. Similarly, depth peeling can be used to account for varying densities beneath the surface, such as bone or muscle, to give a more accurate scattering model.

Figure 10: Using a depth map to simulate scattering. Note that thin areas of the model transmit more light. Image courtesy of NVIDIA

As can be seen in Figure 10, light isn't diffused when passing through object using this technique; back features are clearly shown. One solution to this is to take multiple samples at different points on surface of the depth map. Alternatively, a different approach to approximation can be used, known as *texture-space diffusion.*

As noted at the start of the section, one of the more obvious effects of subsurface scattering is a general blurring of the diffuse lighting. Rather than arbitrarily modifying the diffuse function, diffusion can be more accurately modelled by simulating it in texture space. This technique was pioneered in rendering faces in the Matrix Reloaded [18] as discussed in section 2, but has recently fallen into the realm of real-time techniques.

The method unwraps the mesh of an object using a vertex shader, first calculating the lighting based on the original vertex coordinates. By lighting the unwrapped mesh in this manner, we obtain a 2D image representing the lighting on the object, which can then be processed and reapplied to the model as a normal texture. To simulate diffusion, the light map texture can simply be blurred. Rendering the lighting to a lower-resolution texture in itself provides a certain amount of blurring. The amount of blurring required to accurately model subsurface scattering in skin is still under active research, but performing only a single blur poorly models the true effects [19], as demonstrated in Figure 11.

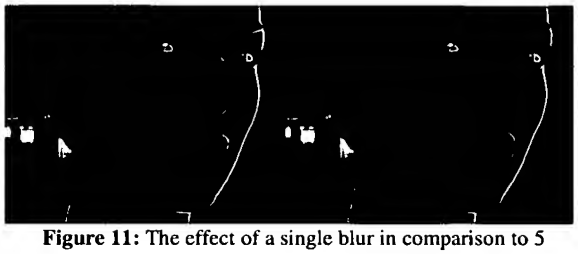


Figure 11: The effect of a single blur in comparison to 5 consecutive blurs, the latter of which models subsurface diffusion more accurately. Images courtesy of NVIDIA.

A major benefit of this method is its independence of screen resolution; shading is performed only once per texel in the texture map, rather than for every pixel on the object. An obvious requirement is thus that the object have a good UV mapping[3]. Similarly, as illustrated in Figure 12, the use of texture space diffusion causes implicit soft shadows, alleviating one of the larger issues in shadowing as discussed in section 3.3.2.

For the foreseeable future, it is likely that subsurface scattering will only be used for skin (particularly ears) as opposed to other materials, due to our expertise at judging humans. However, already sub-surface scattering is seeing a rise in prominence in real-time graphics due to human bias in games, which are reserving greater detail for rendering human characters.

Figure 12: Softened shadows due to texture space blurring. Images courtesy of ATI Research

3.3. Light Transport

Shadows are a common part of the world. The very act of lighting a non-trivial scene will cause areas without illumination. As such, shadows add a great deal of realism, establishing spatial relationships and enhancing the illusion of 3D. Of the methods of generating real-time shadows, only shadow volumes (Crow, 1977) [20] and shadow mapping (Williams, 1978) [21] are in mainstream use. Between them they have inspired numerous variations, additions and enhancements, increasing the quality and efficiency of the base techniques. This segment will review some of the major contributions since the publication of Williams' and Crow's papers, focusing

[3] Each point on the texture must map to only one point of the object.

more on shadow mapping due to the issues that its derivatives have overcome in recent years.

3.3.1. Shadow Volumes

The shadow volume algorithm is a geometry-based technique, in that it creates and modifies geometry to produce the required result. The basic algorithm is a two-pass procedure, consisting of a *shadow determination pass* followed by an *illumination pass*. The determination pass partitions the screen-space into regions of light and shadow. The second pass performs the actual lighting of the scene, illuminating only the lit regions.

The shadow determination pass constructs a *shadow volume* from the geometry in the scene (known as *occluders*), creating a closed polyhedron that extends away from the light to infinity. In Figure 13, the effect of this is to create a triangular frustum. A naïve method of extending this to complex meshes would be to derive a frustum for each triangle in the mesh, requiring eight times the number of triangles in the mesh (two triangles per edge of each triangle plus two caps). To reduce the amount of geometry required, connectivity information can be leveraged to determine the silhouette of the object.

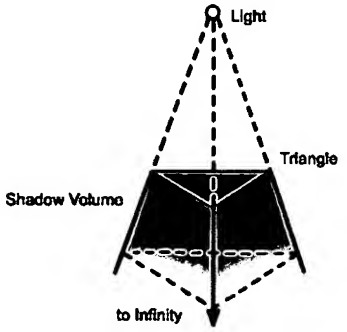


Figure 13: Creating a shadow volume from a triangle. The lines from a point light are extended through the vertices of a triangle to form an infinite triangular frustum, also known as a shadow volume. All geometry that is inside the shadow volume is in shadow.

Silhouette edges are defined [20] as the common edges between forward and back-facing polygons from the light's point of view. The direction of a polygon can easily be decided by the dot product between its normal and the vector from the centre of the polygon towards the light. Once all the polygons are tagged, the connectivity information can be used to identify the neighbouring polygons. An exclusive-or operation between the current and neighbouring polygon's directions can then be used to determine whether the shared edge is a silhouette edge.

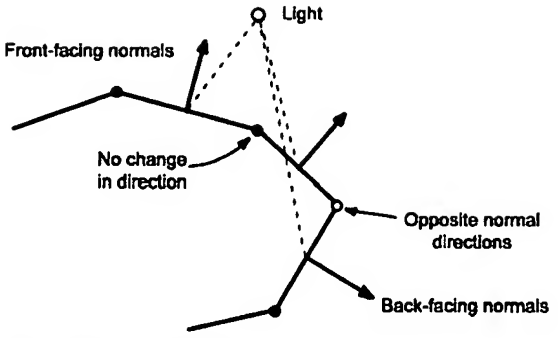


Figure 14: Determining silhouette edges based on neighbouring polygon normals.

The generated volumes are rendered after scene geometry but are prevented from writing to the colour or depth buffers. Instead, the results of z comparisons between the depth buffer and the rasterized shadow volume are used to manipulate the stencil buffer, which can be used to mask writing to the screen at a per pixel level.

A fragment is only in shadow if it is within a shadow volume: in front of the back faces and behind the front faces. If a fragment's stencil buffer value is only incremented when its depth value is less than the back face of the volume and only decremented when in front of the front faces, then the only time the increment/decrement pair will be unbalanced (non-zero) is when the fragment lies within the shadow volume. This extends to multiple volumes, producing non-zero stencil values even for intersecting volumes.

By careful culling, clipping and geometry simplification [22], the main issue of the fill-rate hungry nature of the method can be reduced, in experimental setting up by up to 7 times [23]. It is also possible to create soft shadows using a technique [24] similar to the *smoothie* algorithm presented in the next section, applied to shadow volumes as opposed to shadow maps.

3.3.2. Shadow Mapping

William's approach to shadowing on curved surfaces was to use an image based approach, but light shadow volumes is a dual pass algorithm. The scene is first rendered from the point of view of the light source, in *light space*. At this stage, only the depth values in the form of the z-buffer are computed; no other shading is required. Next, the scene is rendered from the camera view and each visible fragment is transformed into light space, where it is tested for visibility. In practice, the depth values are projected into camera space through projective shadow mapping (see appendix 9.2) allowing the visibility test to simply look up the depth value using the supplied texture coordinates. It is this that led Segal et al. [25] to recognize shadow mapping as a extension of perspective-correct texture mapping.

Shadow mapping utilises the fact that the depth buffer, from the light's point of view, is a pre-computed light visibility test for the light's view frustum, partitioning the volume into lit and shaded regions. The equation to test

whether a fragment is in shadow from the eye's point of view can be written as:

$$p_z \leq shadowMap(p_x, p_y)$$

Where p is a point in light space and *shadowMap* is the *depth buffer* (z-buffer) texture. The test fails if the z value of the incoming point is further from the light source than the value stored in the shadow map texture, and the fragment shaded accordingly. Figure 15 illustrates the concept of shadow mapping by presenting a shadow mapped scene and its equivalent depth buffer.

Figure 15: Shadow mapping: the left image the depth buffer from the light's point of view, the darker values corresponding to a smaller distance. Images courtesy of Haines and Möller.

Shadow mapping allows shadows to be cast from any object onto any surface. It is relatively straightforward to implement and is easily hardware accelerated, due to its reliance on z-buffers and texturing. Most importantly, shadow mapping is not dependant on scene geometry or complexity.

However, shadow mapping in its basic form has serious issues with aliasing, due to variations in sampling frequencies between light and camera space as illustrated in Figure 16. Erroneous self-shadowing can arise due to the quantisation of the z-buffer, as noted in William's original paper. This can be alleviated by the use of a bias value dependent on the z-gradient of the polygon in question. The other drawbacks due to aliasing have been overcome with various levels of success by several modifications of the original algorithm. In the following section, we present the works believed to prove seminal in the coming years.

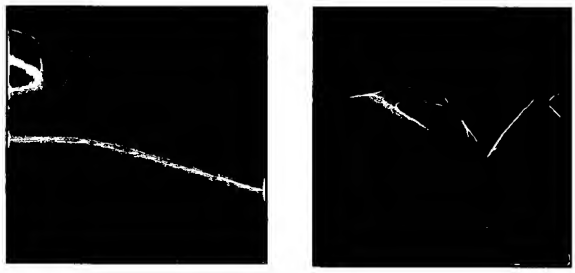


Figure 16: The "duelling frusta" problem shows the variation in sampling frequency between eye and light space; the blue area near the camera (left) requires high frequency sampling, but is only a small portion of the shadow map from the light's view (right). Image courtesy of NVIDIA.

Percentage closer filtering (PCF) [26] is a simple solution to the aliasing problem, a method of blurring the edges between lit and shadowed regions. In general, depth values can't simply be averaged to determine the proportion of shadowing at a fragment. Instead, the depth comparison must be performed first and the Boolean results averaged, as illustrated in Figure 17.

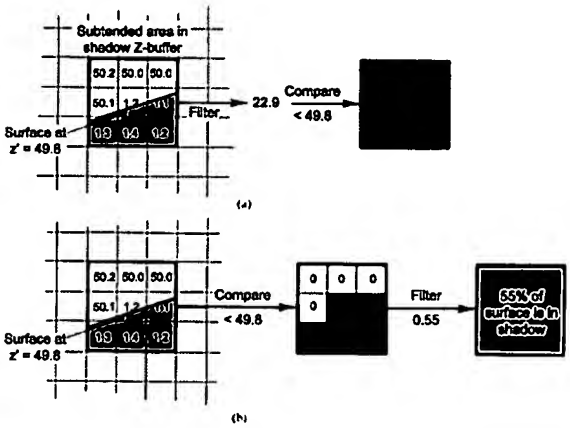


Figure 17: A 3x3 Percentage closer filter kernel, producing a more accurate valuation of the shadowing value in the penumbra than a standard single comparison.

Percentage closer soft shadows (PCSS) [27] takes this a stage further by modifying the size of the PCF kernel, varying the softness of the shadows to approximate the penumbra of a shadow caster. It attempts to estimate the penumbra size by searching on the shadow map near to the current fragment (the *receiver*) for values significantly closer to the light than the current surface, referred to as *blockers*. The blocker depth values are averaged, to be subsequently used in the penumbra size heuristic, which in turn defines size of the PCF kernel to use:

$$w_{penumbra} = \frac{(d_{receiver} - d_{blocker}).w_{light}}{d_{blocker}}$$

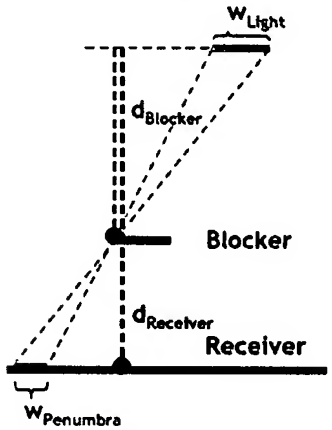


Figure 18: Penumbra size estimation, using the ratio of receiver depth to average blocker depth values. Image courtesy of NVIDIA.

The results of this method are considerably more perceptually accurate than standard PCF: as illustrated in Figure 19, the shadows harden on contact and no aliasing artefacts are evident. Its implicit ability to handle area light sources is also impressive. However, the visual accuracy of the algorithm depends on a large number of blocker search and PCF sample, which are currently quite expensive. As texture lookups continue to reduce in price, PCSS will become more of a contender in realistic soft-shadow rendering.

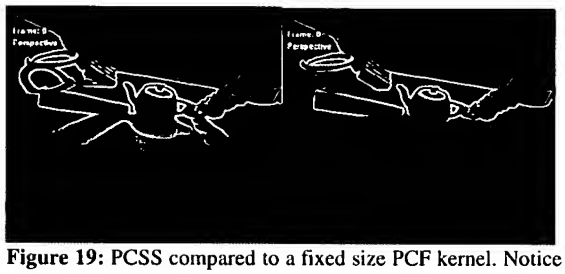


Figure 19: PCSS compared to a fixed size PCF kernel. Notice the hardening of the shadow, accurately reflecting the reducing penumbra size.

Perspective shadow maps (PSM) [28] attempt to alleviate the variation in sampling frequency between light and camera space by gathering the depth values from the point of view of the light after the camera perspective transformation has been applied. A standard shadow map is then generated, rendering a view from the transformed light position to the unit cube. Working in post-perspective space is almost exactly the same as in world space, with the exception of objects behind the viewer. Once transformed by the perspective projection, objects behind the viewer appear on the far side of the infinity plane. If these objects lie within the light view frustum, they may still potentially cast shadows into the view volume and must be accounted for, illustrated in Figure 20. This is handled by using an adjusted perspective projection, moving the viewpoint back until the objects in question lie within the view frustum.

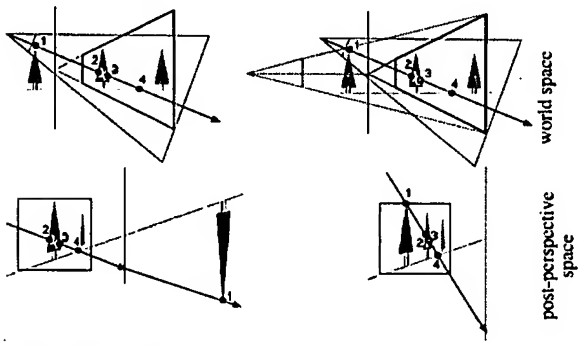


Figure 20: After the transformation, objects behind the viewer (top left) are inverted and appear on the far side of the infinity plane (bottom left). To treat this, the viewpoint is moved back until the object is contained within the view frustum.

As the scene is rendered into the shadow map after the perspective transformation, the "size" of all the shadow map pixels are of uniform size when viewed from camera space, reducing or eliminating aliasing altogether. The effect of this can be seen in Figure 21.

The most time-consuming aspect of the algorithm is calculating the adjusted perspective projection; an extension to PSM, *light space perspective shadow maps* (LiSPSM) [29] avoids this altogether by converting lights into directional lights and constructing a perspective transform $\boldsymbol{P}$ with near and far planes parallel to the lighting direction. After the perspective transformation, the light direction is unchanged and the order of objects is retained, removing the need for scene analysis. As a result, the method runs as fast as uniform shadow mapping.

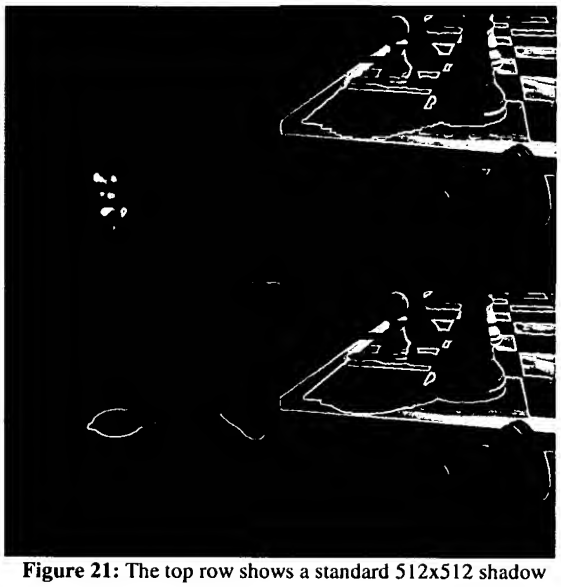


Figure 21: The top row shows a standard 512x512 shadow map and its resultant projection onto the scene. The bottom row shows a shadow map of the same size, gathered after the perspective transform. Note the lack of aliasing in the scene.

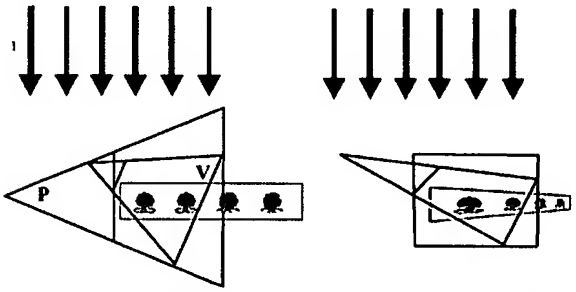


Figure 22: LiSPSM. With a view frustum V, the perspective transform P is defined so the light rays l are parallel to the near and far plane. After perspective transformation, the light direction is unchanged.

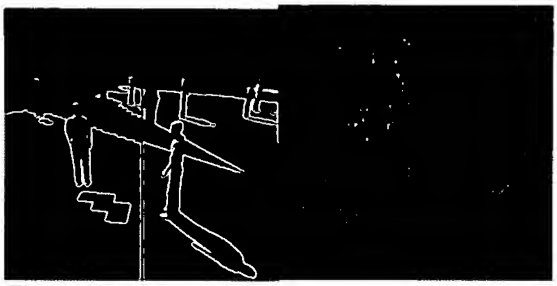


Figure 23: PSM. The left-hand sides of the images were generated with 1024x1024 uniform shadow maps; the right-hand sides were rendered using PSM of the same resolution.

A final method, *rendering fake soft shadows with smoothies*, gained prominence when a related, independently developed method was patented by Crytek [30]. The technique produces results that though not geometrically correct, appear qualitatively like soft shadows with results comparable to ray-tracing. Extra primitives are attached to the silhouettes of objects, creating smooth edges and hiding aliasing artefacts. The method uses a *smoothie buffer* on top of the standard shadow map to retain information about the penumbra (smoothie) locations.

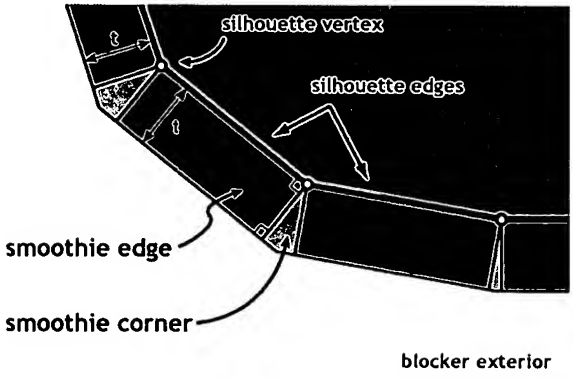


Figure 24: Constructing smoothies on object silhouette edges.

The shadow map is gathered normally, rendering scene occluders into the depth buffer. The occluder's silhouette edges are then calculated (ala shadow volumes) and smoothies constructed on them as shown in Figure 24. The smoothies' depth values are rendered into a *smoothie buffer*, also storing an alpha value interpolated across the smoothie as illustrated in Figure 25. Only one change has to be made from the standard shadow comparison test when rendering the scene: when the point in question lies behind a smoothie from the point of view of the light, the shadow is rendered with the following opacity:

$$\alpha' = \alpha / (1 - d_{blocker} / d_{receiver})$$

Where α is the sampled alpha value from the smoothie buffer. These *remapped alpha* values result in variable width gradients that accurately model the occluders' penumbra.

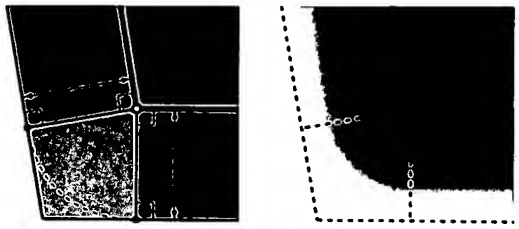


Figure 25: Converting from smoothie geometry to alpha shading

The results of this method, as can be seen in Figure 26, provide realistic soft shadows at very high rendering speeds (see appendix 9.3). The biggest drawback is the smoothie buffer, which requires an additional depth texture and alpha texture on top of the shadow map depth texture.

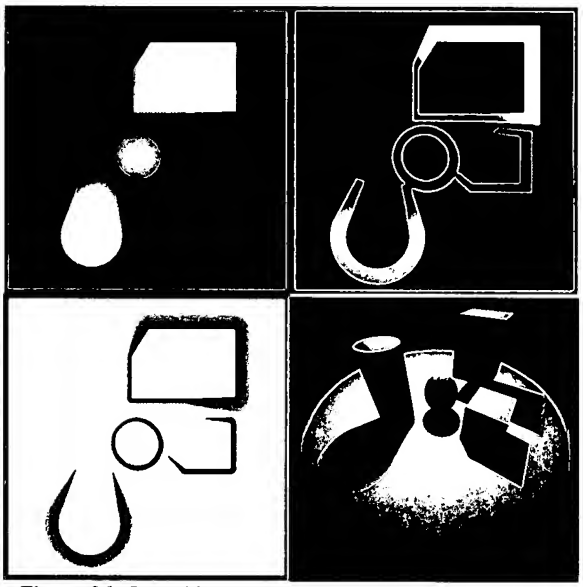


Figure 26: Smoothies. The scene is rendered to a depth map (top left). Smoothies are then attached to the silhouette edges of the objects, and rendered to the smoothie buffer as depth values (top right) and as alpha values (bottom left), interpolated and weighted by the ratio of blocker and receiver distances from the light. The image at the bottom right illustrates the perceptually correct soft shadows rendered with this method.

Shadow mapping has long been considered a more elegant solution than shadow volumes, but its numeric and aliasing issues weighed against it on several occasions, most famously with the choice of shadowing method for Doom 3 [31]. However, the shadow mapping extensions presented in this paper have significantly reduced aliasing, weakening the arguments again it. Additionally, the requirements for creating omni-directional lights are not longer excessive due to various parameterizations of the view frustums [32].

Extensions to shadow volumes have improved performance and increased robustness; the high fill-rate and extra geometry issues mostly resolved and the ability

to add soft shadows leveling the playing field. The deciding factor in the future will therefore be more the ease of implementation and situation context: shadow mapping for scenes with high complexity and occlusion, and volumes for lower complexity.

4. GLOBAL ILLUMINATION

Standard illumination techniques using direct lighting avoid the diffuse inter-reflection component due to its reliance on solving partial differentials. As a result, scenes appear darker than in real life. This can be alleviated with a constant ambient term, ensuring that no surface is completely black. However, this makes areas seem flat and does not account for contact shadows and cracks. One option to combat this is to replace the ambient light with multiple diffuse lights, but the rendering time increases disproportionately for the realism achieved, making it unsuitable for real-time usage. The focus of this section is on exploring alternate techniques that simulate a more accurate ambient term, accounting for illumination with complex light distributions in a real-time context.

4.1. Ambient Occlusion

Surfaces in real life gather incident light from all directions, rather than a small number of directional and positional lights used in standard illumination. As a result a constant ambient term is often inappropriate, as the contributing local environment can fluctuate wildly. Ambient Occlusion (AO) [33] reduces the stark CG look without the use of full Global Illumination by generating a "smart" ambient term that varies over the model, dependant on the range of the external environment it can see. Importantly, this can be run as a pre-process, *baking* it into a texture that can be applied on a model at rendering time as a fast approximation to the diffuse shading term.

The preprocessing stage computes two values for each point on the model: the unoccluded fraction of hemisphere above the point and the average direction of unoccluded incident light. These terms are known as the *accessibility*, and its *bent normal*. Figure 27 illustrates this idea in 2D: an approximation to the average colour of incoming light at P can be made by averaging the light from the cone of unoccluded directions around the bent normal **B**.

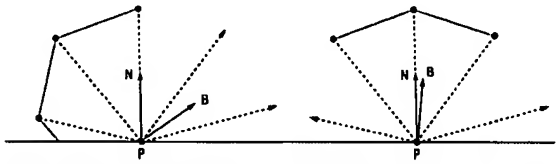


Figure 27: Computing the accessibility and bent normal for a point (left). In some cases, the derived bent normal may point in an occluded direction (right). Images courtesy of NVIDIA.

Figure 28 illustrates a comparison of a complex model on a plane, lit by a simple diffuse shading model and AO accessibility information respectively. Notice on the diffuse shaded model the lack of weight and general unrealistic appearance, exemplified by the excessive brightness of the creature's underside and legs.

The major benefit of this method is its speed; the precomputation of values eliminates overhead at run-time. Similarly, its low requirements mean it can run at peak performance over a wide range of graphics cards.

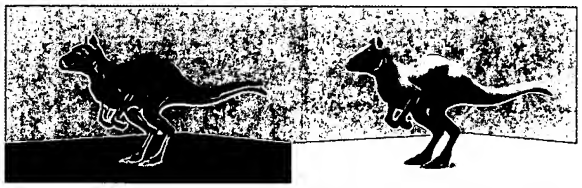


Figure 28: Scene shaded by diffuse model and accessibility information respectively. Note that AO is a local solution, not taking into account the occlusion effects of scene geometry.

Ambient occlusion can be extended, utilising the accessibility information to compute an approximation of the model's appearance within a complex real-world environment, known as *environment lighting*. Although the accessibility factor provides the intensity of the ambient term, it is not modulated by the environment. The bent normal can be used to do a blurred lookup into an environment map to determine the incident ambient light from that direction. Figure 29 demonstrates the effect of environment lighting on a model; it provides an efficient rough approximation to the actual model's appearance in the surroundings depicted in the environment map. The average direction may point in a direction that is actually occluded, as demonstrated in Figure 27, but otherwise the results yielded are usually realistic.

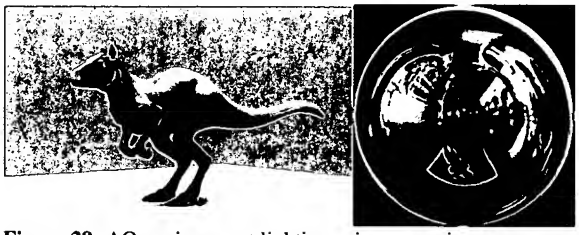


Figure 29: AO environment lighting using an environment map.

As global illumination effects are still relatively hard, *dynamic ambient occlusion* [34] can be used to compute AO in real-time on the GPU by treating a polygonal mesh as a series of disk-shaped elements and calculating the occlusion effect between them. By creating a hierarchical system, approximating multiple elements with single, larger disks, LOD techniques can be used to maintain frame rates with a large number of objects.

A slight modification of this method can introduce an approximation of indirect lighting by factoring in a single bounce, adding a disk-to-disk radiance transfer function. However, this then requires three passes to render: one pass to transfer the reflected or emitted light and two passes to shadow the light.

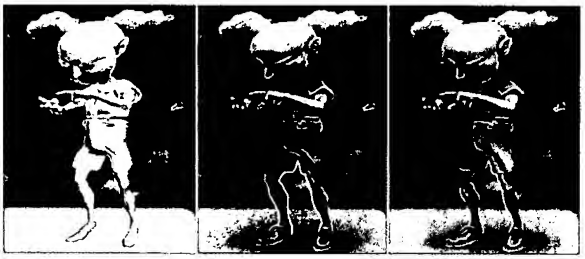


Figure 30: Dynamic AO allows animated meshes to also benefit (middle). Indirect lighting adds an extra level of realism (right).

Ambient occlusion provides a simple and elegant method of increasing the complexity of the ambient term, with low overhead. Static processing produces AO maps that fit well with texture mapping trends (specular; diffuse; normal and height[4]) that the field seems to be following. It also offers the ability to "drop-in" a level of global illumination without the need to redesign the renderer from the ground up.

Dynamic AO is possibly the only technique at present to provide global illumination for dynamic models with real-time frame rates. For this reason it is likely that dynamic ambient occlusion will be a major component of next-generation graphics.

4.2. Radiosity Light Maps

Radiosity simulates the way that reflected light illuminates the surfaces around it. It is an empirical solution using the *finite element method* [35], meaning that the algorithm is iterated until the scene reaches a stable state within some certain threshold. The method is based on the conservation of energy; light scattered by surfaces must be received by others, illuminating them.

Surfaces within the scene are subdivided into *patches*, uniform areas that reemit a proportion of the light they receive and can optionally emit their own light. For each pass, each patch calculates the amount of incident light (see Figure 31) and re-evaluates the level of radiance re-transmitted. This asymptotically moves towards a steady state as energy is dispersed around the scene. The process can be halted either after a certain number of passes or when the change in distribution is within a set bound. Figure 32 shows the gradual distribution of light.

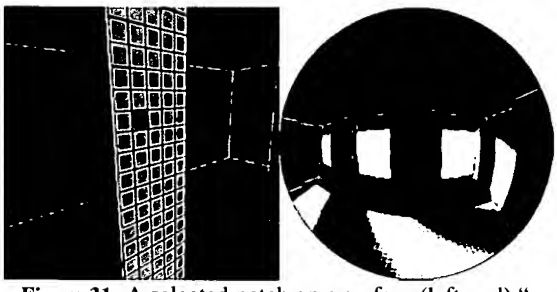


Figure 31: A selected patch on a surface (left, red) "sees" the incident light and updates the amount it emits.

[4] Height maps for parallax mapping

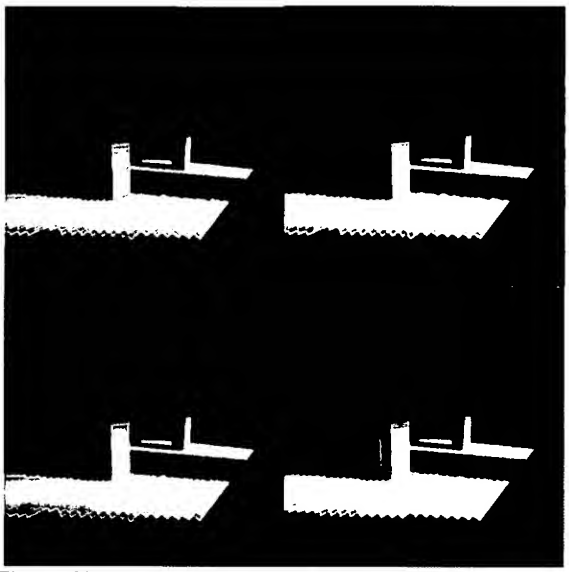


Figure 32: Room illumination after one (top left), two (top right), four (bottom left) and sixteen passes (bottom right). Images by Hugo Elias.

Due to its iterative nature and the number of elements processed radiosity not a real-time algorithm. However, as with ambient occlusion, it can be baked into textures which can then be rendered in real-time. The benefits of radiosity include emergent soft shadows, complex interreflections and colour bleeding, but only for static scenes and lights. A more recent technique, that already has proved to be a seminal paper, allows dynamic lighting using *precomputed radiance transfer*.

4.3. Precomputed Radiance Transfer

Precomputed radiance transfer (PRT) is a technique of pre-processing a scene to capture the complex light interactions between surfaces in low-frequency lighting environments[5]. Based on radiosity, the technique has the same emergent properties, but does not demand a static lighting environment.

Instead of a set radiance value stored in a texture map, PRT stores the response of a surface to its environment as a spherical *radiance transfer function*. This is simply the distribution of incident light on a point from all possible directions (in 3D represented by a sphere, in 2D by a circle). As illustrated in Figure 33, the transfer function initially consists of the environment map, cosine weighted to simulate the contributions made by the environment on a diffuse surface. The distribution is then modulated by a Boolean visibility function, removing contributions from occluded directions. Already, the transfer function is equivalent to an environmentally lit ambient occlusion term in the same location[6].

[5] Areas without sharp changes in illumination

[6] Rather, the integral of the transfer function equals the term.

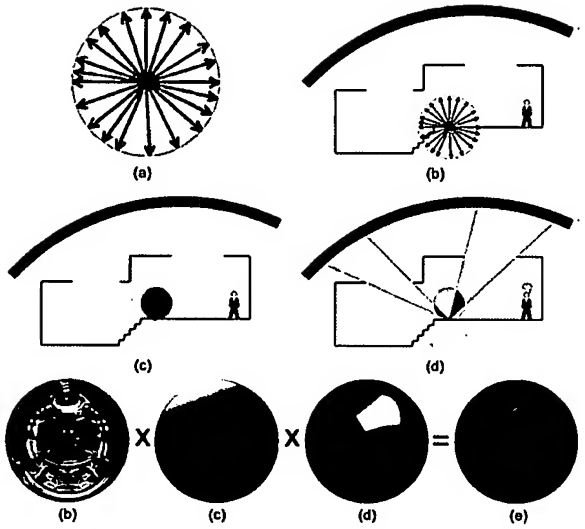


Figure 33: The radiance transfer distribution (a) for a point in a scene (b). The incident light is cosine weighted to approximate a diffuse function (c) and a Boolean visibility test is added, to result in an environmentally lit ambient occluded transfer function (e).

The next stage is to add diffuse interreflections to the transfer function, most easily described as an algorithm:

1. Rays are fired from the point until a triangle surface is hit. Linearly interpolating between the transfer functions at the vertices of the triangle, giving the transfer function with the corresponding amount of light being reflected back towards the shading point.
2. As with direct lighting, this is then scaled by the dot product between the ray and the surface normal (the cosine of the angle), illustrated by Figure 34. Once all the rays are cast the values are normalised, dividing by the number of samples.
3. Once all the rays have been cast, this new set of transfer functions is one set of diffusely interreflected light. Additional bounces use the last computed set of gathered transfer functions as the starting light intensities. The method is repeated until no energy is transferred, or a maximum number of bounces are reached.
4. The globally illuminated transfer function is then the summation of the transfer functions for each bounce and the direct illumination component.

The most important differentiation between this method and the other global illumination methods discussed is the fact that it retains the transfer function rather than a fixed value determined by the lighting conditions at the time. It should be clear that this allows the lighting environment to change at runtime, whilst preserving the interreflections in the scene, as long as the scene remains static.

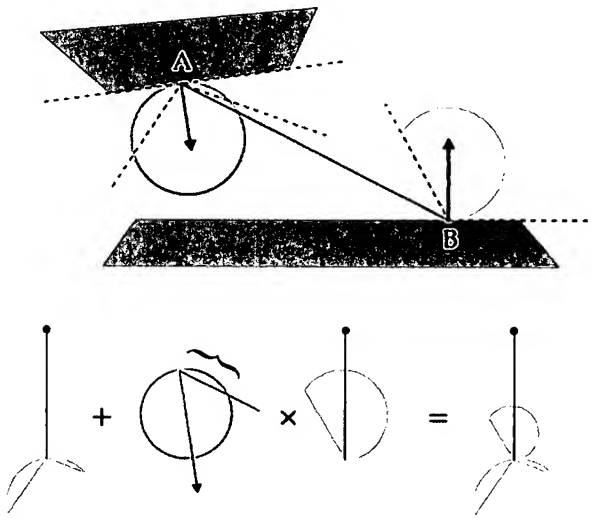


Figure 34: A single sample of diffuse interreflections. The downward facing plane A fires a ray and adds B's scaled transfer function to its own. As a result, A will now be illuminated when a light is overhead.

The transfer function must be compressed to be used in a real-time setting: *spherical harmonics* [36], *Haar wavelets* [37] and *zonal harmonics* [38] are the most popular, though wavelets avoid "side-lobe" artefacts present in spherical harmonics causing light ghosting. They approximate the transfer function as a series of coefficients, which can efficiently be evaluated at run-time as a series of dot products.

Basic PRT is limited to non-local lights and rigid objects that do not move relative to one another. However, extensions [39] and [40] have been developed that allow PRT for positional lights and for deformable objects respectively, though both only handle extremely local effects.

The key to believable lighting that PRT exploits lies with accurately modelling scene interreflections. Soft shadows, so ardently pursued, drop out of the method as an emergent property as do indirect shadowing effects[7]. Colour bleeding dramatically creates effective moods with the minimum of intervention. These components are essential in the future of realistic real-time graphics, placing PRT and its derivatives in a strong position to lead global illumination. Already, the DirectX SDK contains a PRT engine to allow relatively simple integration into the next generation of games.

Currently, few global illumination techniques can perform in real-time. However, the state-of the art is a moving target and a number of non-real time techniques such as [41] and [42] will hopefully make the transition in the near future. Meanwhile, PRT has already started replacing light maps and other static lighting methods in games such as

[7] Indirect shadows are caused by the obscuration of light coming from all directions.

Halo 2, whilst ambient occlusion has begun its inception for dynamic objects.

5. COMPARISONS & RESULTS

The most important aspect of the techniques presented in this report is also the most difficult to measure; the amount of increase in realism achieved due to each method. Obviously it is impossible to assign an absolute value, so it was decided to evaluate the importance of each algorithm as a ranked position.

A Cornell box was rendered using each method and placed on a website. Users were then presented with two randomly chosen pictures and asked to select the more realistic. An affirmation voting-style system, based on the Condorcet criterion, was determined the overall ranking of the images, with the following results:

1. Radiosity / PRT
2. Ambient Occlusion
3. Normal Mapping
4. Parallax Mapping
5. Subsurface Scattering
6. Soft-Shadows using Smoothies
7. LiSPSM / Shadow Volumes
8. Shadow Mapping
9. Phong Shading
10. Gouraud Shading
11. Flat Shading

The rankings conformed to expectations: global illumination methods ranked the highest, demonstrating the importance of indirect lighting to the solidity of the scene. Texture effects rated higher than shadowing methods, users valuing surface material information over the positional hints gained from shadows. Initially, Phong shading did not beat Gouraud shading by a significant margin, due to a lack of curved surfaces in the scene. They were re-rendered with the addition of a sphere to better illustrate the difference.

6. OPTICAL EFFECTS

More generally known as post-processing, these are not part of in the rendering equation. All forms of sensing equipment have inherent limitations, modifying the received image. These translate into effects like lens flares, exposure and motion blur.

High dynamic range lighting (HDR) [43] is a method of retaining greater precision lighting information about a scene in order to more accurately represent both dark and light areas. With low dynamic range (LDR), bright areas are clipped to white at each stage, causing errors and artefacts to accumulate and capping rendering output to the surface material albedo. HDR uses floating point numbers throughout the rendering pipeline, storing more information than will ultimately be displayed on screen. The resulting image must be *tone mapped*, selecting a range of intensities and scaling them to fit the LDR range of the screen, equivalent to setting the exposure level on a camera.

Glow, otherwise known as bloom [44], provides important cues about atmosphere and brightness. Glows and halos are caused by atmospheric scattering in the atmosphere and in our eyes from bright sources. In graphics, the maximum intensity of light is capped. As a result, brightness hints are limited to the glow effects surrounding sources. Glow is easily achieved, by solely rendering the specular and emissive elements of a scene to a texture, blurring and compositing the result.

Post processing effects admittedly add style to an image, but they are often overused and serve to distract from more basic problems of render quality. In the coming years, they will become more subtle - more accurate – in line with the increased realism that comes with the techniques presented in this paper.

7. CONCLUSIONS

Existing games are a series of compromises: dynamic lighting is balanced against pre-computed; direct illumination is effective but reflectance models are simple, and shadows add weight but are unrealistic. Each limitation forces artists to adapt their skills to cope, with the effect that the end product suffers. The next generation of real-time graphics is set to put more control in the hands of artists, with accurate multi-layered materials, qualitative soft shadows and dynamic global illumination.

Although global illumination methods add indirect light to a scene, for the foreseeable future they will not be able to integrate lighting for moving and deformable objects with static geometry. As a result, direct illumination and shadowing will remain a large part of realistic rendering in real-time graphics.

Shadow mapping and volumes have become increasing robust, producing realistic soft shadows that render efficiently. The choice of implementation is now more due to ease of implementation rather than a significant gain in efficiency, thus biasing towards shadow mapping.

Materials are becoming more complex, due to the freedom presented by fragment shaders. Physically correct BRDFs are coming into use, accurately modelling light interaction to produce effects such as sub-surface scattering. Additionally, shaders are being used to add simulate geometric detail to surfaces, the trend gradually proceeding from normal mapping to parallax mapping, towards complete ray-tracing.

The field of real-time graphics is not lacking in vision: globally illuminating completely dynamic scenes is still the holy grail of research; caustics are currently not largely ignored for real-time application as are the effects of atmospherics. If the computer graphics previously was focused on rendering phenomenologically correct scenes, the future of graphics is in physically correct rendering.

8. REFERENCES

[1] M. F. Cohen, J. Wallace and P. Hanrahan, *Radiosity and Realistic Image Synthesis*, Academic Press Professional, 1993.

[2] D. Elder, *Advanced Lighting and* Shading, Course Technology PTR, More OpenGL Game Programming, 2005, pp. 235-49.

[3] R. L. Cook, K. E. Torrance, *Reflectance Model for Compuer Graphics*, Computer Graphics, Vol. 15, No. 3, 1981.

[4] J. Blinn, *Models of Light Reflection for Computer Synthesized Pictures*, Computer Graphics, Vol. 11, No. 2, 1977.

[5] A. Ngan, F. Durand and W. Matusik, *Experimental Validation of Analytical BRDF Models*, SIGGRAPH '04, Technical Sketch, 2004.

[6] B. T. Phong, *Illumination for Computer Generated Pictures*, Communications of the ACM, Vol. 18, No. 6, 1975.

[7] J. Owens, *Streaming Architectures and Technology Trends*, GPU Gems 2, Addison-Wesley Professional, 2005, p. 467.

[8] M. Oren and S. K. Nayar, *Generalization of Lambert's Reflectance Model*, ACM Press, Proceeding of Computer Graphics, 1994, pp. 239-46.

[9] M. Ashikhmin and P. Shirley, *An Anisotropic Phong BRDF Model*, Journal of Graphics Tools, Vol. 5, No. 2, 2000, pp. 25-32.

[10] T. Kaneko, et al., *Detailed Shape Representation with Parallax Mapping*, ICAT, 2001.

[11] T. Walsh, *Parallax Mapping with Offset Limiting*, Infiniscape, Tech Report, 2003.

[12] Morgan McGuire and Max McGuire, *Steep Parallax Mapping*, I3D 2005 Poster. Available online at: http://www.cs.brown.edu/research/graphics/SteepParallax/

[13] W. Donnelly, *Per-Pixel Displacement Mapping with Distance Functions*, GPU Gems 2, Addison-Wesley Professional, 2005, pp.123-36.

[14] NVidia Geforce 7 Series GPU Specifications, available at: http://www.nvidia.com/object/7_series_techspecs.html

[15] A. Krishnaswamy and G. V. G. Baronoski, *A Biophysically-based Spectral Model of Light Interaction with Human Skin,* Blackwell Publishing, Computer Graphics Forum, Vol. 23, No. 3, 2004.

[16] S. Green, *Real-time Approximations to Subsurface Scattering*, GPU Gems, Addison-Wesley Professional, 2004, pp. 263-278.

[17] Z. Nagy and R Klein, *Depth-Peeling for Texture-based Volume Rendering*, 11th Pacific Conference on Computer Graphics and Applications (PG'03), 2003, p. 429.

[18] G. Borshukov and J.P. Lewis, *Realistic human face rendering for "The Matrix Reloaded"*, ACM Press, Computer Graphics, 2005.

[19] E. d'Eon, *Advanced Skin Rendering*,GDC 2007, http://developer.download.nvidia.com/presentations/2007/gdc/Advanced_Skin.pdf

[20] F. C. Crow, *Shadow Algorithms for Computer Graphics*, ACM Press, Computer Graphics, Vol. 11, No. 2, 1977, pp. 242-48.

[21] L. Williams, *Casting Curved Shadows on Curved Surfaces*, ACM Press, Computer Graphics, 1978, pp. 270-74.

[22] M. McGuire, J. F. Hughes, C. Everitt, *Fast, Practical and Robust Shadows*, Technical report, NVIDIA. 2003. http://developer.nvidia.com

[23] B. Lloyd, J. Wendt, N. Govindaraju and D. Manocha, *CC Shadow Volumes*, ACM Press, SIGGRAPH 04 Sketches. 2004, p. 146.

[24] U. Assarsson, M. Dougherty, M. Mounier, T. Akenine-Möller, *An Optimised Soft Shadow Volume Algorithm with Real-time Performance*, Eurographics Association, Proceedings of SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware, 2003, pp. 33-40.

[25] M. Segal et al., *Fast shadow and lighting effects using texture mapping, ACM Press,* Computer Graphics, Vol. 26, No. 2, 1992, pp. 283-91.

[26] W. Reeves, D. Salesin and R. Cook, *Rendering Antialiased Shadows with Depth Maps*, ACM Press, Computer Graphics, Vol. 21. No. 4, 1987, pp. 283-91.

[27] R. Fernando, *Percentage-closer Soft Shadows*, ACM Press, Proceedings of SIGGRAPH, 2005.

[28] M. Stamminger and G. Drettakis, *Perspective Shadow Maps*, ACM Press, Proceedings of International Conference on Computer Graphics, 2002.

[29] M. Wimmer, D. Scherzer, W. Purgathofer, *Light Space Perspective Shadow Maps*, Eurographics Symposium on Rendering, 2004.

[30] M. Corbetta, Patent: *Method, computer program product and system for rendering soft shadows in a frame*, 2001/022133, US 6903741, Crytek, 2001.

[31] J. Carmack: Email to private list, May 23, 2000. http://developer.nvidia.com/.

[32] S. Barbec, T. Annen, H. P. Seidel, *Shadow Mapping for Hermispherical and Omnidirectional Light Sources*, Proceedings of Computer Graphics International, 2002, pp. 397-408.

[33] M. Pharr and S. Green, *Ambient Occlusion*, GPU Gems, Addison-Wesley Professional, 2004, pp. 279-92.

[34] M. Bunnell, *Dynamic Ambient Occlusion and Indirect Lighting*, GPU Gems 2, Addison-Wesley Professional, 2005, pp. 223-33.

[35] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method*, Elsevier, 5th edition, 2000.

[36] R. Green, *Spherical Harmonic Lighting: The Gritty Details*, Game Developers' Conference, 2003.

[37] R. Ng, R. Ramamoorthi, R. Hanrahan, *Triple product wavelet integrals for all-frequency relighting*, ACM TOG (SIGGRAPH 04), Vol. 23, No. 3, 2004, pp. 475-85.

[38] E. Whittaker and G. Watson, A Course in Modern Analysis, Cambridge University Press, 4th edition, 1990.

[39] A. W. Kristensen, T. Akenine-Möller and H. W. Jensen, *Precomputed Local Radiance Transfer for Real-time Lighting Design*, ACM Press, Proceedings of ACM SIGGRAPH 05, 2005.

[40] P. P. Sloan, B. Luna and J. Snyder, *Local, Deformable Precomputed Radiance Transfer*, ACM Press, SIGGRAPH 05, 2005, pp. 1214-24.

[41] P. Gautron, J. Krivanek, K. Bouatouch and S. Pattanaik, *Randiance Cache Splatting: A GPU-Friendly Global Illumination Algorithm*, ACM Press, SIGGRAPH 05 Sketches, 2005.

[42] B. D. Larsen, N. J. Christensen, *Simulating Photon Mapping for Real-Time Applications*, Proceedings of GRAPHITE, 2003, pp. 1-11.

[43] A. Reinot, *High Dynamic Range Lighting and Rendering*, Course Technology PTR, More OpenGL Game Programming, 2005, pp. 267-84.

[44] G. James and J. O'Rorke, *Real-Time Glow*, GPU Gems 2, Addison-Wesley Profession, 2005, pp. 343-62.

9. APPENDIX

9.1. Calculating Texture Space

Multiple coordinate systems are used in 3D graphics, including object space, world space and image space. Normal mapping relies on another system, known as *texture space*, related to the texture coordinates associated with each vertex in a model. A surface normal is by definition orthogonal to a polygonal surface at a vertex. The surface *tangent* and *binormal* together define a 2D coordinate system parallel to the surface at a given vertex and whose axes point along the texture axes *s* and *t* respectively. The *binormal* vector can be calculated as the cross product of the surface normal and tangent, deriving the texture space coordinate system [N, T, B]. Multiplying a vector in object space by this matrix will then convert it into texture space.

9.2. Projective Texture Mapping

Projective texturing is a method of mapping a texture onto an object such that the texture is *projected* onto the scene, as shown in ***Figure 35***. This is accomplished by the use of inbuilt texture co-ordinate generation facilities.

Figure 35: Two views of a projective texture, courtesy of NVIDIA.

OpenGL provides access to the *eye coordinates* stage of the transformation pipeline (see Figure 36) for texture coordinate generation, so incoming fragments are specified in eye space as opposed to object space. Eye coordinates are subsequently transformed by the *texgen* planes[8] and the *texture matrix*, both of which are user specified.

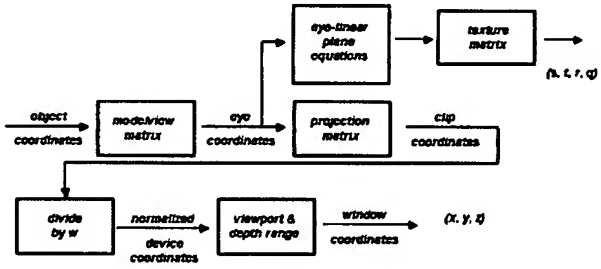


Figure 36: The OpenGL Transformation Pipeline, courtesy of NVIDIA.

To produce projective texturing, coordinates in object space $(\mathbf{x_e,y_e,z_e,w_e})$ must be transformed into *homogeneous* texture coordinates **(s,t,r,q)** such that **(s/q, t/q)** are coordinates in *real space* and **r/q** is the z-distance from the origin in light space. This can be calculated as follows:

$$[s,t,r,q]^T = SP_{light}L^{-1}V[x_e,y_e,z_e,w_e]^T$$

Equation 1

Here, the inverse of the view matrix, **V**, shifts the point from eye space to world space, as shown in Figure 37. Note the convention of defining the forward transforms as going to world space.

[8] Four plane equations that for the purposes of the project can be viewed as a 4x4 matrix.

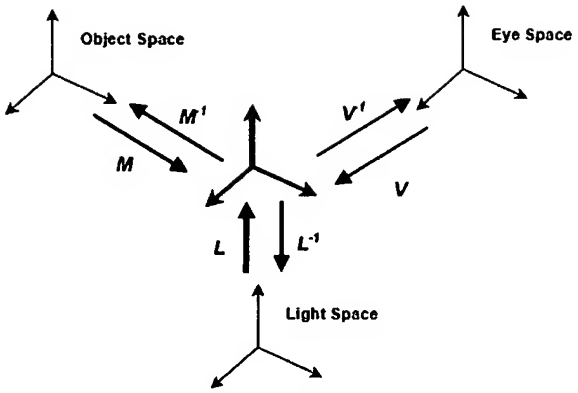


Figure 37: Basic transformations involved in projective texturing, courtesy of NVIDIA

Applying matrix $\mathbf{L}^{-1}$ moves the system to light space and $\mathbf{P}_{light}$ sets the projective matrix for the light frustum. However, the resultant coordinate system has the range [-1, 1], so to convert to a valid texture space [0, 1] a scale-bias matrix must be applied:

$$\begin{bmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 2

Looking at Figure 36, the two transformations applied to eye coordinates produce (**s,t,r,q**). The $\mathbf{SP}_{light}\mathbf{L}^{-1}\mathbf{V}$ transformation can thus be spread between the texgen planes $\mathbf{E}_{po}$ and texture matrix **T**:

$$[s,t,r,q]^T = TE_{po}[x_e, y_e, z_e, w_e]^T$$

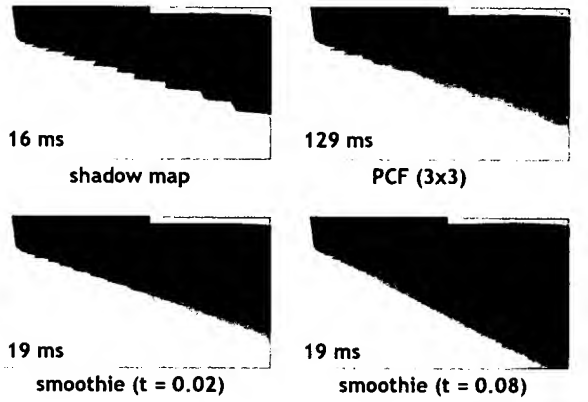
$$TE_{po} = SP_{light}L^{-1}V$$

Equation 3

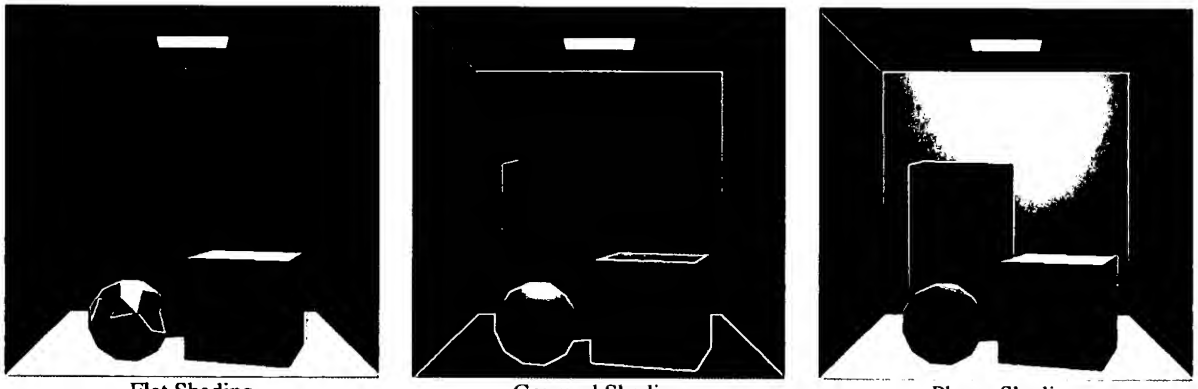
For *eye linear* texgen planes, OpenGL will multiply the eye coordinates by the inverse of the modelview matrix in effect at the time the planes were set. Setting the modelview matrix to $\mathbf{V}^{-1}$ will thus result in the elimination of **V** from Equation 3.

9.3. Soft-Shadow Mapping Comparisons

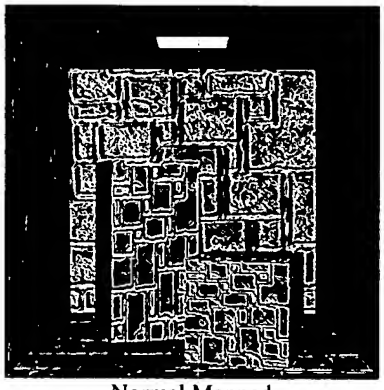
Size	512x512	1024x1024
Control	16ms	17ms
Smoothies	19ms	22ms (t = 0.2) 24ms (t = 0.8)
PCF (3x3)	129ms	142ms



9.4. Cornell Boxes

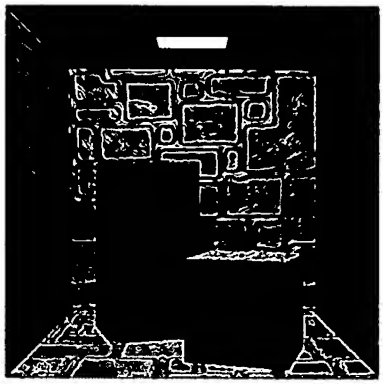


Flat Shading | Gouraud Shading | Phong Shading



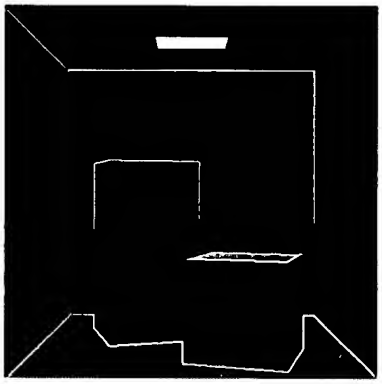
Normal Mapped

Normal Mapped + Texture

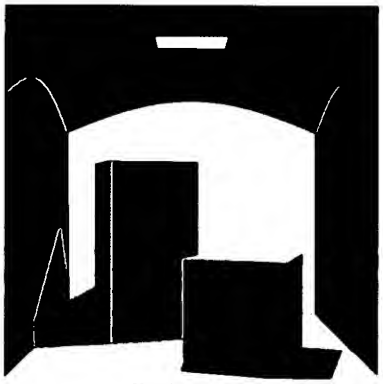


Parallax Mapped

Parallax Mapped + Texture

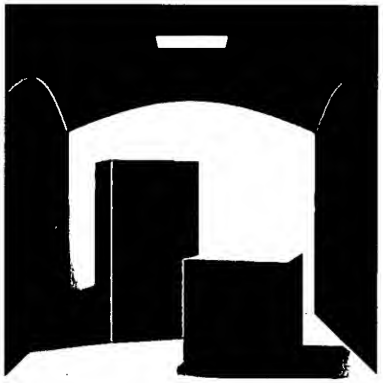


Shadow Volumes

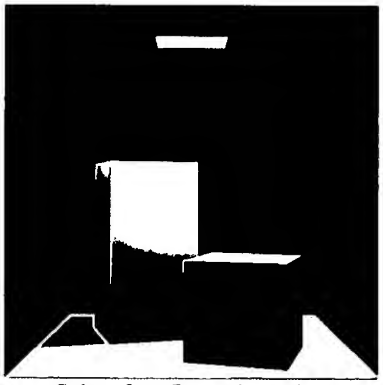


Shadow Map

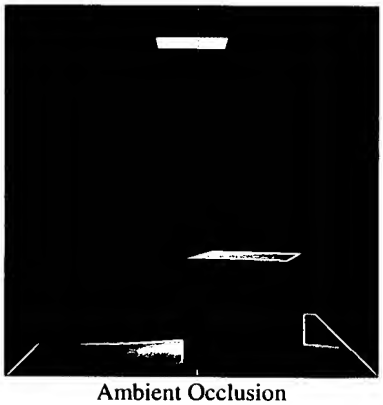
Light space perspective shadow map



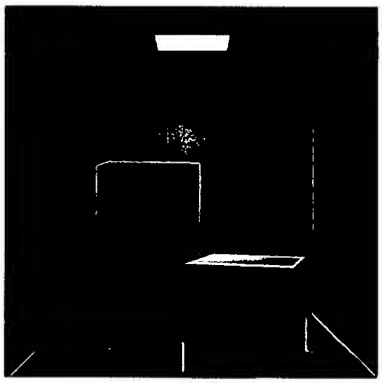
Soft Shadows using Smoothies



Subsurface Scattering using Depth Maps



Ambient Occlusion



Radiosity